1.(*décrit une liste de symbols et leur traduction en bits*)
```
type encoding = (Symbol.t * Bits.t) list
```

2.(*Fonction d'incrémentation d'entier*)
```
let incr : int -> int
```

3.(*ajout d'un bit par la gauche*)
```
let rec aG : Bit.t -> Bits.t -> Bits.t
```

(*TRANSLATION *)

4.(*bourrage de bits à zero pour des codes binaires de meme taille*)
```
let rec compl : int -> Bits.t -> Bits.t
```

5.(*Nombre de bits pour écrire un entier en binaire*)
```
let rec nBits : int -> int
```

6.(*Codage d'une liste de symboles d'alphabet, de meme taille que nb en binaire *)
```
let rec code : symbols -> int -> int -> encoding
```

7.(*fonction associant un symbole à un code binaire*)
```
let build_encoding : Alphabet.t -> encoding
```

8.(*trouve le code associé au symbole*)
```
let rec associated_symbol : Symbol.t -> encoding -> Bits.t
```

9.(*change un symbole en sa traduction en bits *)
```
let rec bits_in_symbols_list: (Symbol.t list) -> Bits.t ->  (Symbol.t list)
```

10.(*change une liste de symbole en une liste de leur traduction en bits *)
```
let rec replaceBand : Symbol.t list -> encoding -> Bits.t list
```

11.(*change une liste de bits en liste de symbols *)
```
let rec bitBand_to_Band : Bits.t list -> Symbol.t list
```

12.(*traduit une bande de symbols normaux en bande de symbols de {B,D} *)
```
let encode_band : encoding -> Band.t -> Band.t
```

13.(*traduit la liste de bandes de symbols normaux en bande de symbols de {B,D} *)

```
let rec encode_with : encoding -> Band.t list -> Band.t list
```

(* REVERSE TRANSLATION *)

14.(*change une liste de bits sous forme de symboles en bits *)

```
let rec decode_bits : int -> Symbol.t list -> Bits.t
```

15.(*décale la liste de symbole du nombre de décalage donnés*)

```
let rec decale_list : int -> Symbol.t list -> Symbol.t list
```

16.(*renvoi le symbol associé aux bits*)

```
let rec associated_Bits :  Bits.t -> encoding -> Symbol.t
```

17.(*change une liste de bits sous forme de symbole en une liste de symboles correspondant*)

```
let rec symbit_to_bits : int -> encoding -> Symbol.t list -> Symbol.t list
```

18.(*nombre de bits de la liste de bits donnée*)

```
let rec nombreMaxBits: Bits.t -> int
```

19.(*renvoit la liste de symvol de l'encoding*)

```
let rec symblist : (Symbol.t*Bits.t) -> Symbol.t
        =fun (s,b) -> s
```

20.(*decodage d'une bande de bits sous forme de symboles en bande de symbole*)

```
let decode_band : encoding -> Band.t -> Band.t
```

21. (*decodage de bandes de bits sous forme de symboles en bandes de symbole*)

```
let rec decode_with : encoding -> Band.t list -> Band.t list
```

```
# let alphabet = Alphabet.make [B;Z;U];;
let bandana = Band.make alphabet [U;U;Z;U];;
let bandana2 = Band.make alphabet [U;B;Z;U];;
val alphabet : Alphabet.alphabet =
  {symbols = [B; Z; U]; symbol_size_in_bits = 2}
# val bandana : Band.band =
  {left = []; head = U; right = [U; Z; U]; color = Color.COL
"LightGray";
   alphabet = {symbols = [B; Z; U]; symbol_size_in_bits = 2}}
# val bandana2 : Band.band =
  {left = []; head = U; right = [B; Z; U]; color = Color.COL
"LightGray";
   alphabet = {symbols = [B; Z; U]; symbol_size_in_bits = 2}}

# let encodementation = build_encoding alphabet ;;
val encodementation : encoding = [(B, [B; B]); (Z, [D; B]); (U, [B;
D])]

# encode_with encodementation [bandana;bandana2];;
- : Band.Band.t list =
[{left = []; head = D; right = [B; D; B; B; D; D; B];
  color = Color.COL "LightGray";
  alphabet = {symbols = [B; D]; symbol_size_in_bits = 1}};
 {left = []; head = D; right = [B; B; B; B; D; D; B];
  color = Color.COL "LightGray";
  alphabet = {symbols = [B; D]; symbol_size_in_bits = 1}}]

# encode_band encodementation bandana;;
- : Band.Band.t =
{left = []; head = D; right = [B; D; B; B; D; D; B];
 color = Color.COL "LightGray";
 alphabet = {symbols = [B; D]; symbol_size_in_bits = 1}}

# bandana;;
- : Band.band =
{left = []; head = U; right = [U; Z; U]; color = Color.COL
"LightGray";
 alphabet = {symbols = [B; Z; U]; symbol_size_in_bits = 2}}

# decode_band  encodementation (encode_band encodementation bandana
);;
- : Band.Band.t =
{left = []; head = U; right = [U; Z; U]; color = Color.COL
"LightGray";
 alphabet = {symbols = [B; Z; U]; symbol_size_in_bits = 2}}
```